CJML to Neural Network

Yimeng Ren

2019/10/22

Contents

1 Dataset Preview

1.1 Data Introduction

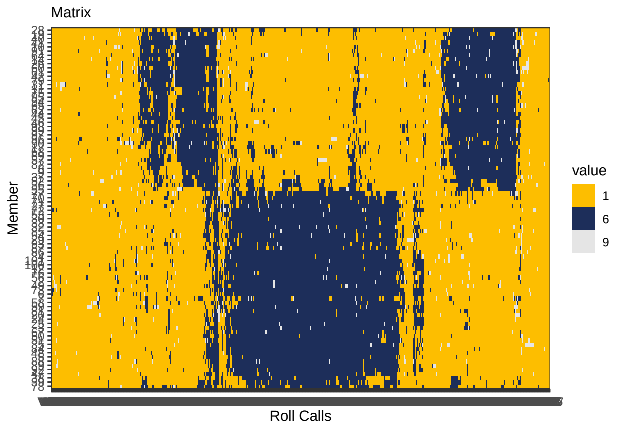
This project use data from 108th Senate Roll Call Data, which contains person-item matrix: 675 Roll Calls, 101 Members.

- Note: there are some members who vote for few calls, and some calls only voted by few members, which are viewd by outliers and excluded from the data for analysis.

After removing the person record whose not voting rate is more than 5%, the data used for modelling contains response from 86 people and 675 items.

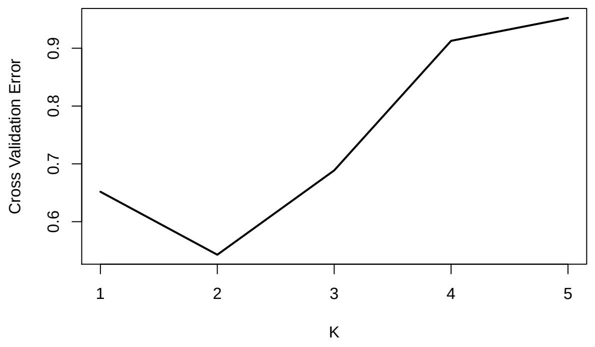
1.2 Heat Matrix

- Yellow elements indicates that a specific member vote for the specific call, while the dark blue indicates voting against.



From the ordered heat map, the preliminary judgement of K is 2. Next we test it by 5-fold cross validation.

1.3 Cross Validation for choosing K



Accroding to the result of cross validation in CJML paper, the propriate K is set as 2.

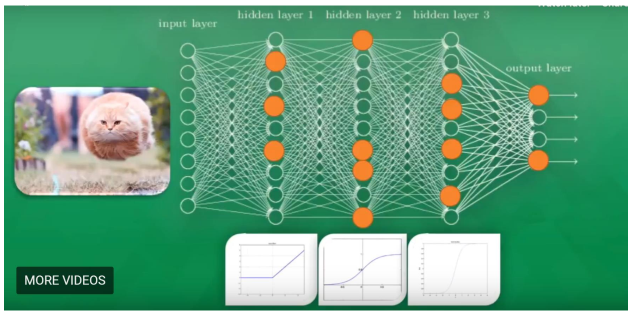


Figure 1: Neural Network Trianing Process

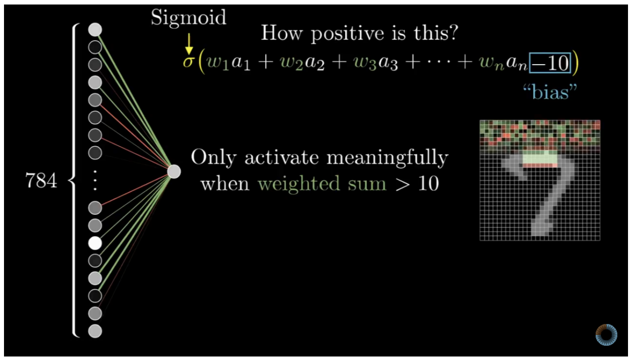

Sigmoid
How positive is this?
$\sigma(w_1a_1 + w_2a_2 + w_3a_3 + \cdots + w_na_n \boxed{-10})$
"bias"
784
Only activate meaningfully when weighted sum > 10

Figure 2: Activate Function to Calculate Weighted Sum

2 Neural Network Introduction

2.1 Training Process and Structure

- Each layer has an activation function, whose result would be passed on to next layer (see Figure 1).

- Take the hand-witten picture as an example, the biggger activation in one pixel is, the more likely it would be activated.
- If the predict error could not be tolerated, the weights and bias in this network would be updated, some appearing sensitive while others insensitive.

2.2 Parameters and Weights

Neurons with parameters and connections with weights, calculate inner product (weighted sum, see Figure 2)

$$\sigma\left(\begin{bmatrix} w_{00} & w_{01} & \cdots & w_{0n} \\ w_{10} & w_{11} & \cdots & w_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k0} & w_{k1} & \cdots & w_{kn} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right) = a^{(1)} = \sigma(Wa^{(0)} + b)$$

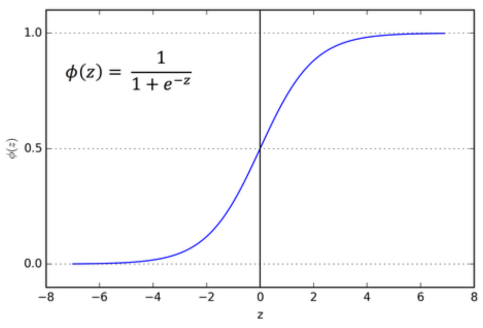


Figure 3: Sigmoid Function

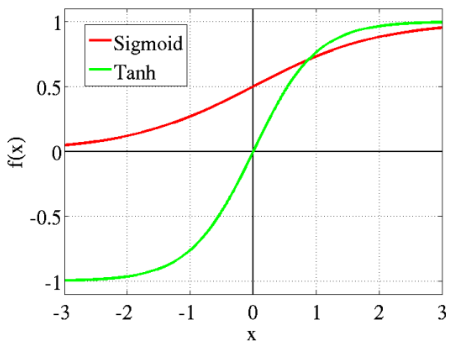


Figure 4: Tanh Function

2.3 Activate Function

2.3.1 Sigmoid

- It is especially used for models where we have to predict the probability as an output.
- The softmax function is a more generalized logistic activation function which is used for multiclass classification.

2.3.2 Tanh

- The tanh function is mainly used classification between two classes.

2.3.3 Relu (Rectified Linear Unit)

- The ReLU is the most used activation function in the world right now.Since, it is used in almost all the convolutional neural networks or deep learning.

2.4 Propagate Backwards

- Use backpropagation to find the individual weights' contribution to the error function (see Figure 6).
- The descent of each parameter represents how much would the influence on loss function be if this paramater is updated.

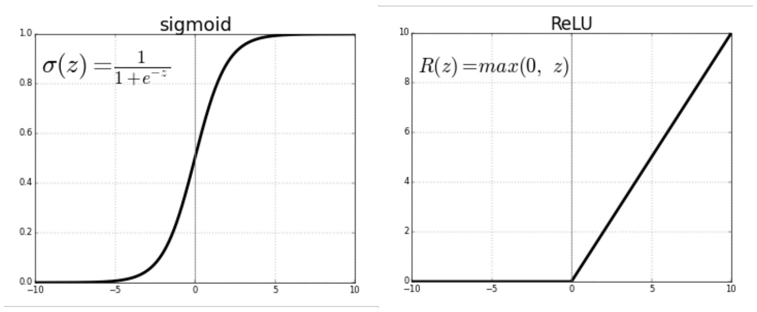


Figure 5: ReLU Function

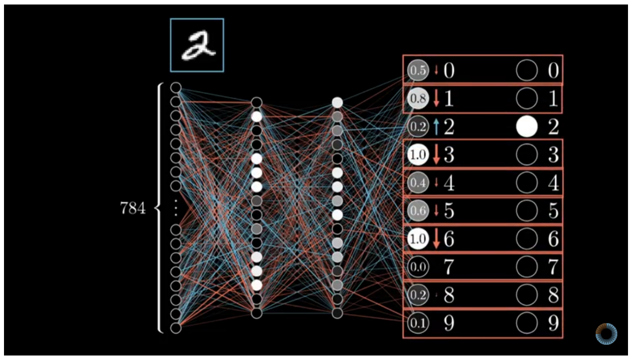


Figure 6: Back Propagation

Take the example of one specific training sample (In fact, all the training samples should be repeated):

- the size of nudges should be proportional to how far it is to the target value.
- Take one of the output neuron of this training sample as an example:
  - change b
  - change w_i in proportion to a_i
  - change a_i in proportion to w_i

3 CJML to Neural Network

As the computation method in CJML paper shows, a similar neural network model is designed with multiple inputs and embedding layers in Figure 7.

3.1 Convergence

Using 500 by 2000 simulation person-item matrix to tell the convergence of this network model. The result shows that no matter MSE or cross entropy, this model could converge to the baseline (Figure 8).

3.2 Batch Size

Since the loss decreases as the batch size increases in simulation data (Figure 9), we choose to set a large batch size (set as 64) for real data, a 86 by 675 matrix.

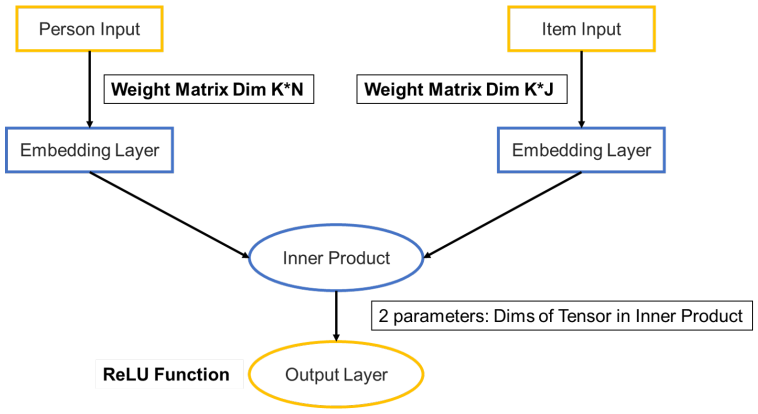


Figure 7: CJML to Neural Network

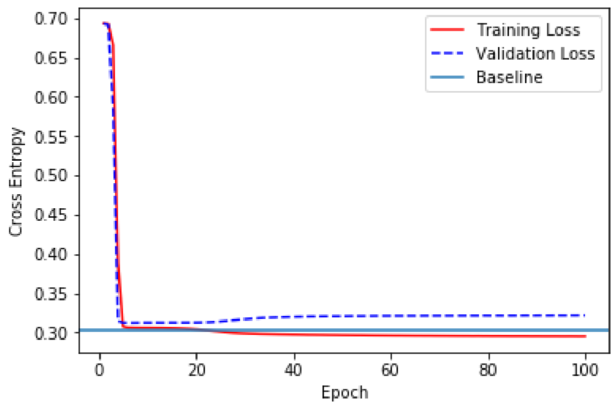


Figure 8: Convergence by Epoch

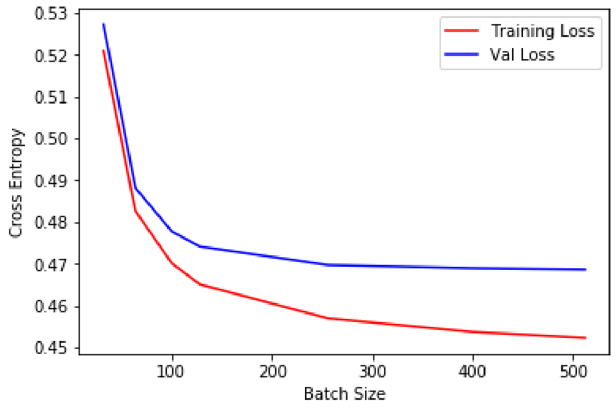


Figure 9: Loss Trend by Batch Size (Simulation data)

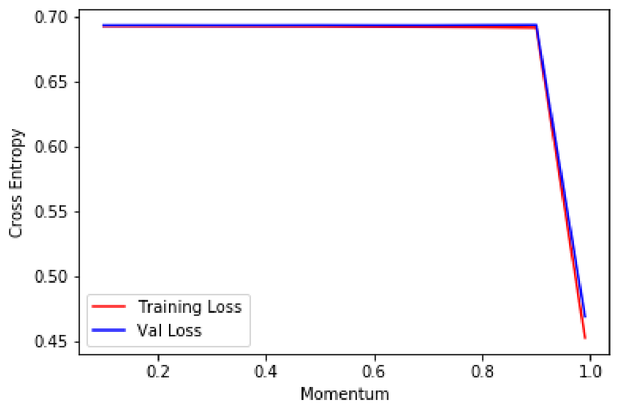


Figure 10: Loss Trend by Momentum (Simulation data)

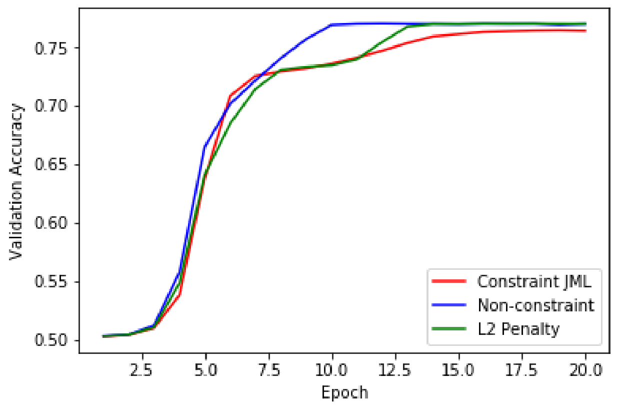


Figure 11: Constraint Comparison on Simulation Data

3.3 Momentum

The loss decreases significantly when momentum over 0.9, therefore set as 0.99 (Figure 10).

3.4 Constraint: L2 Penalty and CJML

- Without constraint, the model appears over-fitting during training.
- Implement both constraint JML and L2 Penalty, it seems that the latter performs better in validation accuracy in **simulation data** (Figure 11).
- Similar performance of calidation accuracy in **real data**, but the L2 penalty seems more efficient (earlier increase).

4 Custom Optimizer: Alternating Minimization

The model training processes above are based on SGD optimizer since there is no alternating opitimizer according to the CJML paper in kears. Therefore, I wrote a custom alternating mininization optimizer for this specific person-item parameters problem (Detailed code in supplementary code file).

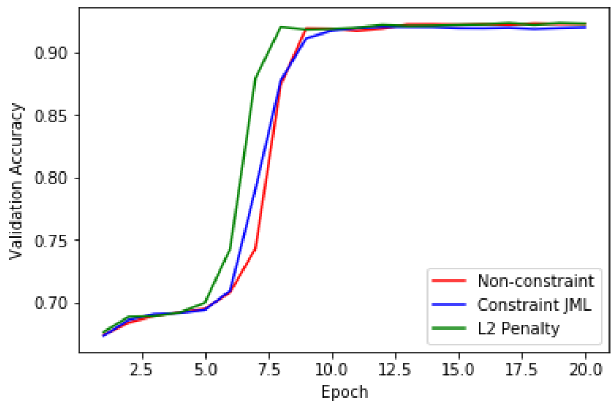


Figure 12: Constraint Comparison on Real Data

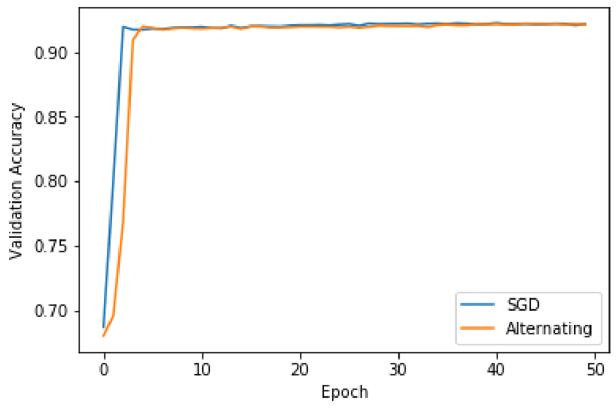


Figure 13: Constraint Comparison on Simulation Data

The main solution is to setback part of the parameter matrix after the update in one iteration with SGD, which generates the same new parameters as alternating minimization does.

4.1 Compare alternating optimizer and SGD

Compared with SGD, the validation accuracy with alternating optimizer increases slower, but still reaches nearly the same max accuracy after 5 epochs (Figure 13).

4.2 Compare prediction of CJML and Neural Network on real data

I choose the same 20% data as test set from the real data (108th roll calls data), and perform CJML estimation in R with `K = 2`, output the prediction accuracy of 0.9178. Compare it with the Neural Network result as Figure 14 shows.

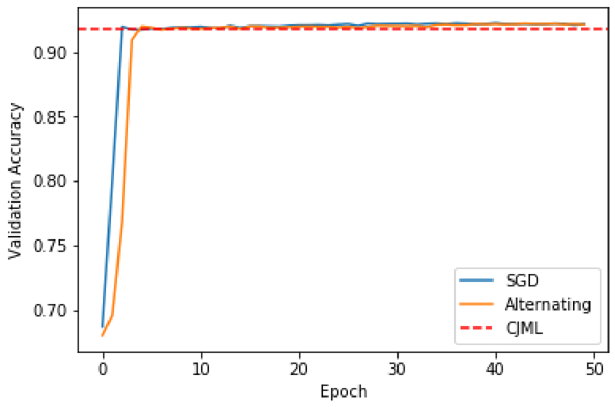


Figure 14: Constraint Comparison on Simulation Data